项目开发规范

一、目的

对于代码，首要要求是它必须正确，能够按照程序员的真实思想去运行；第二个的要求是代码必须清晰易懂，使别的程序员能够容易理解代码所进行的实际工作。在软件工程领域，源程序的风格统一标志着可维护性、可读性，是软件项目的一个重要组成部分。而目前还没有成文的编码风格文档，以致于很多时候，程序员没有一个共同的标准可以遵守，编码风格各异，程序可维护性差、可读性也很差。通过建立代码编写规范，形成开发小组编码约定，提高程序的可靠性、可读性、可修改性、可维护性、可继承性和一致性，可以保证程序代码的质量，继承软件开发成果，充分利用资源，使开发人员之间的工作成果可以共享。

本文在参考原来已有的编码风格的基础上，描述了一个基于 公司 PB 的项目风格，力求一种统一的编程风格，并从整体编码风格、代码文件风格、函数编写风格、变量风格、注释风格等几个方面进行阐述。（这些规范并不是一定要绝对遵守，但是一定要让程序有良好的可读性）

项目开发环境

必须保证开发环境和测试环境以及授控库环境的一致性.

二、整体编码风格

1、缩进

缩进建议以一个TAB键为单位。Tab Size 为在不同的系统中占位不同，建议在VI中设置 set tabstop 为 4。预处理语句、全局数据、标题、附加说明、函数说明、标号等均顶格书写。语句块的"{"、"}"配对对齐，并与其前一行对齐，语句块类的语句缩进建议每个"{"、"}"单独占一行，便于匹对。

2、空格

原则上变量、常量数据和函数在其类型，修饰名称之间适当空格并据情况对齐。关键字原则上空一格，如：if （...）等。运算符的空格规定如下："->"、"["、"]"、"++"、"--"、"~"、"!"、"+"、"-"（指正负号）、"&"（引用）等几个运算符两边不加空格（其中单目运算符系指与操作数相连的一边），其它运算符（包括大多数二目运算符和三目运算符"?:"两边均加一空格，在作函数定义时还可据情况多空或不空格来对齐，但在函数实现时可以不用。不论是否有括号，对语句行后加的注释应用适当空格与语句隔开并尽可能对齐。个人认为此项可以依照个人习惯决定遵循与否。

3、对齐

原则上关系密切的行应对齐，对齐包括类型、修饰、名称、参数等各部分对齐。另每一行的长度不应超过屏幕太多，必要时适当换行，换行时尽可能在","处或运算符处，换行后最

好以运算符打头，并且以下各行均以该语句首行缩进，但该语句仍以首行的缩进为准，即如其下一行为“{”应与首行对齐。

变量定义最好通过添加空格形成对齐，同一类型的变量最好放在一起。如下例所示：

```
int       value;
int       result;
int       length;
double    amt;
double    amount;
```

个人认为此项可以依照个人习惯决定遵循与否。

4、空行

不得存在无规则的空行，比如说连续十个空行。程序文件结构各部分之间空两行，若不必要也可只空一行，各函数实现之间一般空两行，由于每个函数还要有函数说明注释，故通常只需空一行或不空，但对于没有函数说明的情况至少应再空一行。函数内部数据与代码之间应空至少一行，代码中适当处应以空行空开，建议在代码中出现变量声明时，在其前空一行。

5、注释

注释是软件可读性的具体体现。程序注释量一般占程序编码量的 20%，软件工程要求不少于 20%。程序注释不能用抽象的语言，类似于"处理"、"循环"这样的计算机抽象语言，要精确表达出程序的处理说明。例如："计算总金额"、"打印文件体"等。避免每行程序都使用注释，可以在一段程序的前面加一段注释，具有明确的处理逻辑。

注释必不可少，但也不应过多，不要被动的为写注释而写注释。以下是四种必要的注释：

A. 标题、附加说明。

B. 函数等的说明。对几乎每个函数都应有适当的说明，通常加在函数实现之前，在没有函数实现部分的情况下则加在函数原型前，其内容主要是函数的功能、目的、算法等说明，参数说明、返回值说明等，必要时还要有一些如特别的软硬件要求等说明。公用函数、公用类的声明必须由注解说明其使用方法和设计思路，当然选择恰当的命名格式能够帮助你把事情解释得更清楚。

C. 在代码不明晰或不可移植处必须有一定的说明。

D. 及少量的其它注释，如自定义变量的注释、代码书写时间等。

注释有块注释和行注释两种，分别是指："/**/"和"//"建议对 A 用块注释，D 用行注释，B、C 则视情况而定，但应统一，至少在一个单元中 B 类注释形式应统一。具体对不同文件、结构的注释会在后面详细说明。

6、页宽

页宽应该设置为 80 字符。源代码一般不会超过这个宽度, 并导致无法完整显示, 但这一设置也可以灵活调整. 在任何情况下, 超长的语句应该在一个逗号或者一个操作符后折行. 一条语句折行后, 应该比原来的语句再缩进一个 TAB 键.

三、代码文件风格

所有的 c(*.ec,*.c) 文件都必须遵守如下的样式规则：

. 头文件生成

对于同一主控下的程序，尽量包含用 get_all 工具来生成头文件，避免用手工制作的头文件. 对于私有的头文件建议只放在当前主控目录下,不必要放到$HOME/incl 目录下

. 文件头部注释

文件头部注释主要是表明该文件的一些信息，是程序的总体说明，可以增强程序的可读性和可维护性。文件头部注释一般位于 文件最前面。要求至少写出文件名、创建者、创建时间和内容描述。
例如:

```
/**
 * ProgramName:   pbmain.ec
 * SystemName:   OFP PreBranch+ System
 * Version: 1.0
 * OS & Environment: IBM/AIX4.3,Sybase 11.0,ANSI/C
 * Description
 * History
 * Date              Position      Author      Description    Address
* 2002.12.05      福建邮政    yihui            Modify         GuangZhou
 */
```

. 库文件

接下来的是包含应有的库函数和头文件。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "ftpapi.h"

EXEC SQL include "pbvardef.h";
```

EXEC SQL include "sys/sys_errno.h";

接下来是全局变量的定义以及宏定义等

EXEC SQL BEGIN DECLARE SECTION;
staticPKG_481014　pkg481014;/* ATM 上传包 */
staticPKG_481015　pkg481015;/* ATM 下传包 */
staticPKG_488001　pkg488001;/* 电银上下传包 */
EXEC SQL END DECLARE SECTION;

#define SIGNCODE"28109"

接下来是函数声明：
void print_pkg481012( );

四、函数编写风格

. 函数的命名

通常，函数的命名也是以能表达函数的动作意义为原则的，一般是由动词打头，然后跟上表示动作对象的名词，各单词的首字母应该大写。另外，还有一些函数命名的通用规则。如取数，则用 get 打头，然后跟上要取的对象的名字；设置数，则用 set 打头，然后跟上要设的对象的名字；类似的规则还有很多，需要程序员多读优秀的程序，逐渐积累经验，才能作出好的函数命名。

. 函数注释

系统函数，不必太多的注释和解释；

对于自行编写的函数，若是系统关键函数，则必须在函数实现部分的上方标明该函数的信息，格式如下：

/**
* Function：
* Action：
* Inputs：
* Outputs：
* Return：
*/

希望尽量遵循以上格式。

五、符号风格

. 总体要求

对于各种符号的定义，都有一个共通点，就是应该使用有实际意义的英文单词或英文单词的缩写，不要使用简单但没有意义的字串，尽可能不使用阿拉伯数字，更切忌使用中文拼音的首字母。如这样的名称是不提倡的：Value1,Value2,Value3,Value4 …。

例如：
file(文件),code(编号),data(数据),pagepoint(页面指针), faxcode(传真号) ,address(地址),bank(开户银行),……

. 神秘的数

首先要说什么是神秘的数。我们在程序里经常会用到一些量，它是有特定的含义的。例如，现在我们写一个薪金统计程序，公司员工有 50 人，我们在程序里就会用 50 这个数去进行各种各样的运算。在这里，50 就是"神秘的数"。为什么称它为神秘呢？因为别的程序员在程序里看到 50 这个数，不知道它的含义，只能靠猜了。

在程序里出现"神秘的数"会降低程序的可读性，应该尽量避免。避免的方法是把神秘的数定义为一个常量。注意这个常量的命名应该能表达该数的意义，并且应该全部大写，以与对应于变量的标识符区别开来。例如上面 50 这个数，我们可以定义为一个名为 NUMOFEMPLOYEES 的常量来代替。这样，别的程序员在读程序的时候就可以容易理解了。

六、性能

在写代码的时候，从头至尾都应该考虑性能问题。这不是说时间都应该浪费在优化代码上，而是我们时刻应该提醒自己要注意代码的效率。比如：如果没有时间来实现一个高效的算法，那么我们应该在文档中记录下来，以便在以后有空的时候再来实现她。

不是所有的人都同意在写代码的时候应该优化性能这个观点的，他们认为性能优化的问题应该在项目的后期再去考虑，也就是在程序的轮廓已经实现了以后。